Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch:2023-2028	**Due date:**

Experiment 3: Email Spam or Ham Classification using Naive Bayes, KNN, and SVM

1 Aim:

To design and implement classification models using Naive Bayes variants and K-Nearest Neighbors (KNN) algorithms to accurately classify emails as spam or ham. Additionally, to evaluate and compare their effectiveness using multiple performance metrics.

2 Libraries used:

- Numpy
- Pandas
- Matplotlib
- Scikit-learn
- Seaborn

3 Objective:

- To preprocess the email dataset by cleaning text data, vectorizing features, and splitting the data for training and testing.
- To implement Naive Bayes classifiers (Bernoulli, Multinomial, Gaussian) and KNN classifiers, tuning parameters such as k-value.
- To measure and compare model performance using accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC, enabling informed model selection.

4 Implementation Code:

```
#IMPORTS
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time

from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Binarizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report, roc_curve, auc
)
```

```
# --------------------------
# 1. Load Dataset
# --------------------------
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/spambase_csv.csv')
print("Shape:", df.shape)
print(df.head())
```

```
Mounted at /content/drive
Shape: (4601, 58)
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0            0.00               0.64           0.64           0.0
1            0.21               0.28           0.50           0.0
2            0.06               0.00           0.71           0.0
3            0.00               0.00           0.00           0.0
4            0.00               0.00           0.00           0.0

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0           0.32            0.00              0.00                0.00
1           0.14            0.28              0.21                0.07
2           1.23            0.19              0.19                0.12
3           0.63            0.00              0.31                0.63
4           0.63            0.00              0.31                0.63

   word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0             0.00            0.00  ...           0.00          0.000
1             0.00            0.94  ...           0.00          0.132
2             0.64            0.25  ...           0.01          0.143
3             0.31            0.63  ...           0.00          0.137
4             0.31            0.63  ...           0.00          0.135

   char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0            0.0          0.778          0.000          0.000
1            0.0          0.372          0.180          0.048
2            0.0          0.276          0.184          0.010
3            0.0          0.137          0.000          0.000
4            0.0          0.135          0.000          0.000

   capital_run_length_average  capital_run_length_longest  \
0                       3.756                          61
1                       5.114                         101
2                       9.821                         485
3                       3.537                          40
4                       3.537                          40

   capital_run_length_total  class
0                       278      1
1                      1028      1
2                      2259      1
3                       191      1
4                       191      1

[5 rows x 58 columns]
```

```
# --------------------------
# 2. Check & handle missing
# --------------------------
```

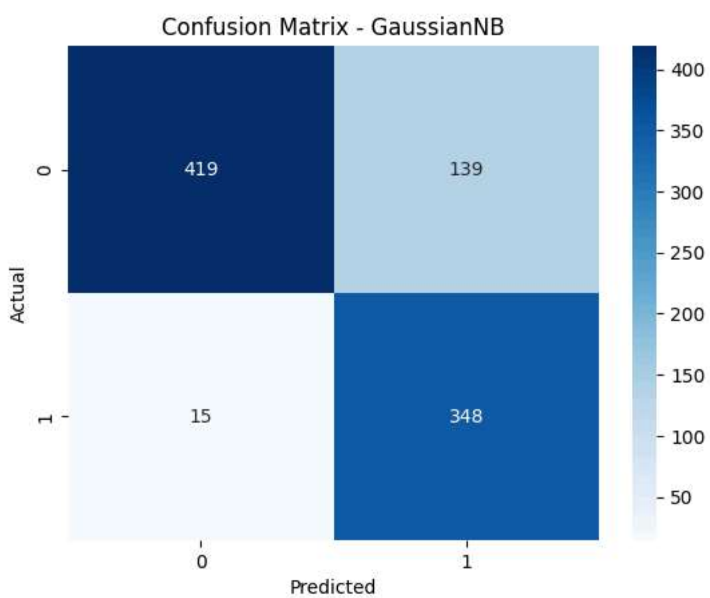
```
print("\nMissing values:", df.isna().sum().sum())
df = df.dropna()
```

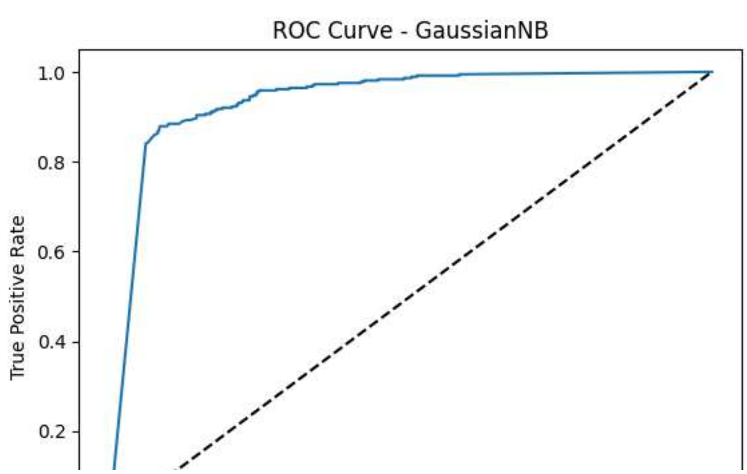
```
Missing values: 0
```

```
# --------------------------
# 3. Separate features & label
# --------------------------
X = df.iloc[:, :-1]
y = df.iloc[:, -1]



# --------------------------
# 4. Scale features
# --------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)



# --------------------------
# 5. Train-Test split
# --------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)


# --------------------------
# Utility functions
# --------------------------
def evaluate_model(model, X_train, X_test, y_train, y_test):
    start = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start
    y_pred = model.predict(X_test)
    metrics = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'Train Time (s)': train_time
    }
    return model, y_pred, metrics

def plot_confusion(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {title}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

def plot_roc(model, X_test, y_test, title):
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:,1]
    else:
        y_score = model.decision_function(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{title} (AUC={roc_auc:.2f})')
    plt.plot([0,1], [0,1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {title}')
    plt.legend()
    plt.show()

# --------------------------
# 6. Naive Bayes Variants
# --------------------------
nb_models = {
    'GaussianNB': Pipeline([('scaler', StandardScaler()), ('clf', GaussianNB())]),
    'MultinomialNB': Pipeline([('minmax', MinMaxScaler()), ('clf', MultinomialNB())]),
```

```
    'BernoulliNB': Pipeline([('binarizer', Binarizer(threshold=0.0)), ('clf', BernoulliNB())])
}

nb_results = {}
for name, model in nb_models.items():
    m, y_pred, metrics = evaluate_model(model, X_train, X_test, y_train, y_test)
    nb_results[name] = metrics
    print(f"\n{name} Report:\n", classification_report(y_test, y_pred))
    plot_confusion(y_test, y_pred, name)
    plot_roc(m, X_test, y_test, name)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print(accuracy_score(y_test, y_pred))
print(precision_score(y_test, y_pred))
print(recall_score(y_test, y_pred))
print(f1_score(y_test, y_pred))

nb_df = pd.DataFrame(nb_results).T
print("\nNaive Bayes Comparison:")
print(nb_df)
```

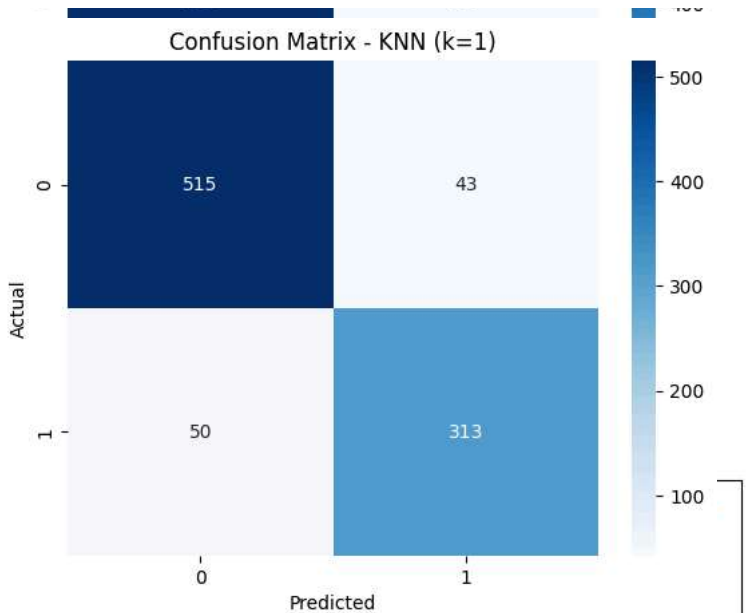
```
GaussianNB Report:
               precision    recall  f1-score   support

           0       0.97      0.75      0.84       558
           1       0.71      0.96      0.82       363

    accuracy                           0.83       921
   macro avg       0.84      0.85      0.83       921
weighted avg       0.87      0.83      0.83       921
```

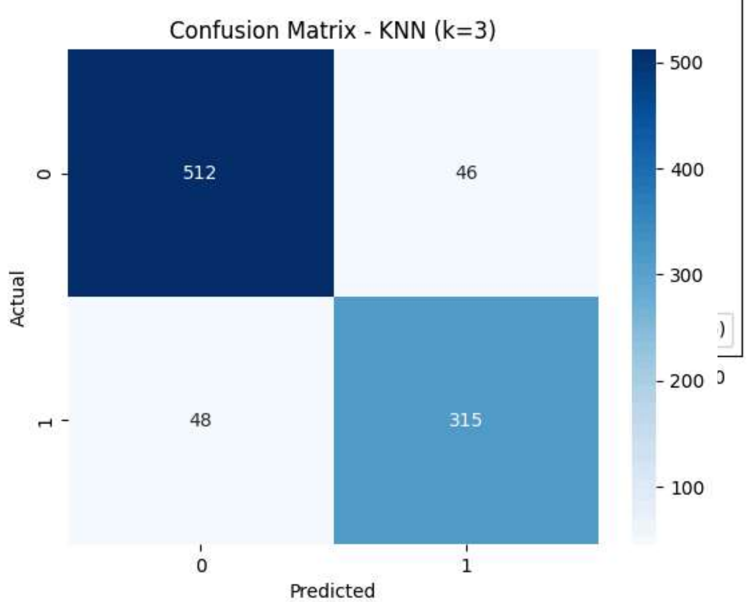





```
# -------------------------
# 7. KNN - different k values
# -------------------------
k_values = [1, 3, 5, 7]
```

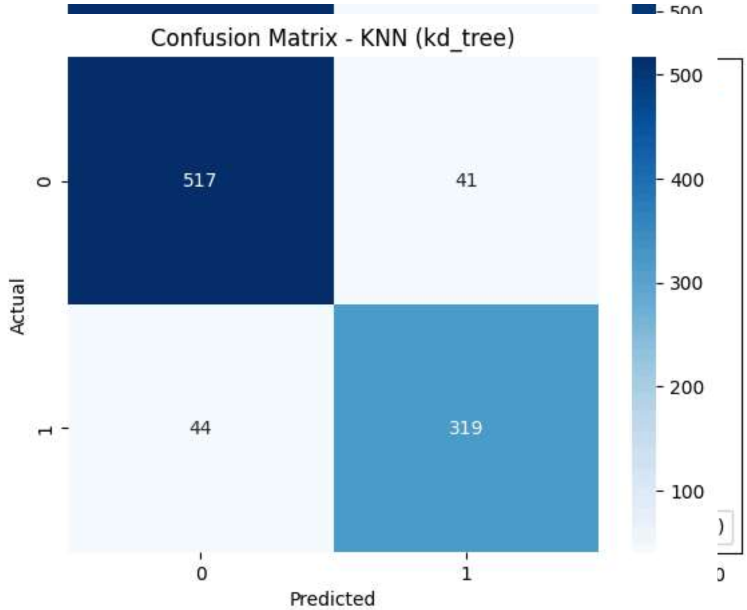
```
knn_results = {}
for k in k_values:
    knn = Pipeline([('scaler', StandardScaler()), ('clf', KNeighborsClassifier(n_neighbors=k))])
    m, y_pred, metrics = evaluate_model(knn, X_train, X_test, y_train, y_test)
    knn_results[f'k={k}'] = metrics
    plot_confusion(y_test, y_pred, f'KNN (k={k})')

knn_df = pd.DataFrame(knn_results).T
print("\nKNN (varying k) Comparison:")
print(knn_df)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print(accuracy_score(y_test, y_pred))
print(precision_score(y_test, y_pred))
print(recall_score(y_test, y_pred))
print(f1_score(y_test, y_pred))
```

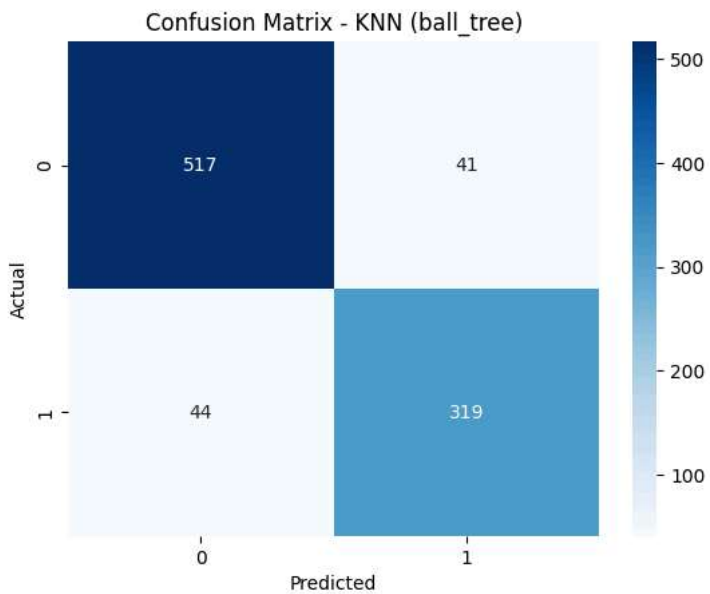





```
# -------------------------
# 8. KNN - KDTree vs BallTree
# -------------------------
tree_results = {}
for algo in ['kd_tree', 'ball_tree']:
```

```
    knn = Pipeline([('scaler', StandardScaler()), ('clf', KNeighborsClassifier(n_neighbors=5, algorithm=algo))])
    m, y_pred, metrics = evaluate_model(knn, X_train, X_test, y_train, y_test)
    tree_results[algo] = metrics
    plot_confusion(y_test, y_pred, f'KNN ({algo})')

tree_df = pd.DataFrame(tree_results).T
print("\nKNN Tree Comparison:")
print(tree_df)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print(accuracy_score(y_test, y_pred))
print(precision_score(y_test, y_pred))
print(recall_score(y_test, y_pred))
print(f1_score(y_test, y_pred))
```

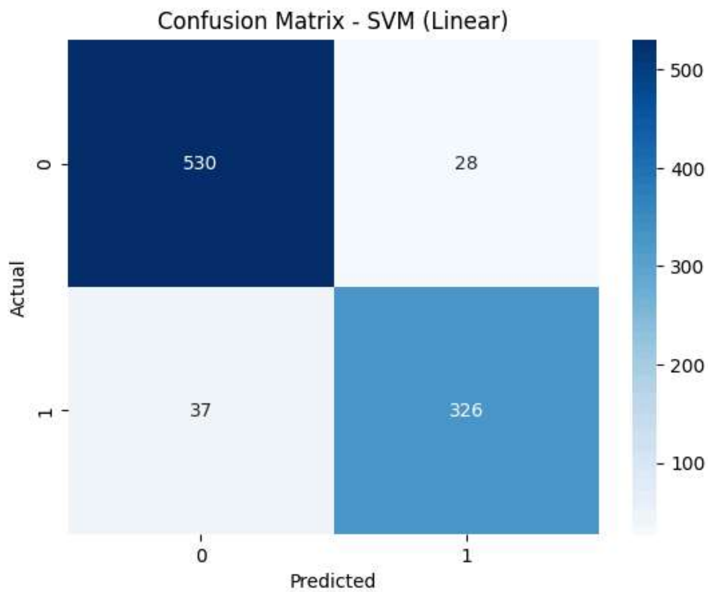


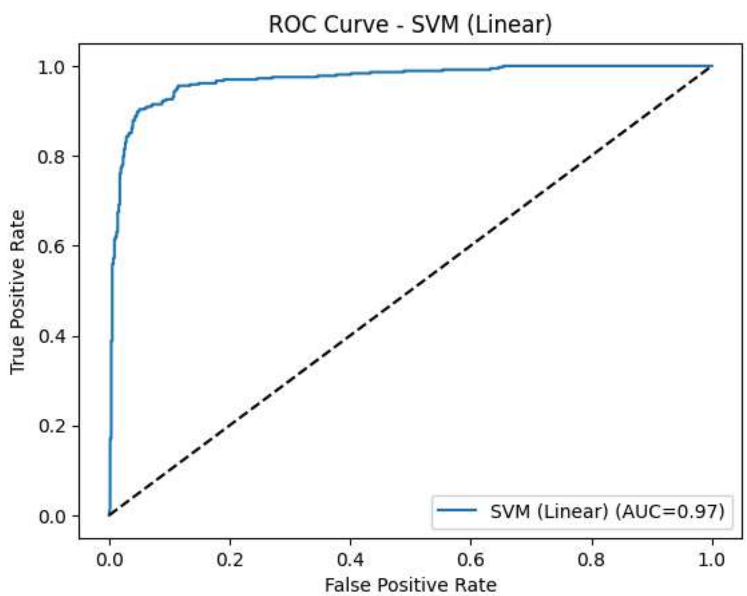


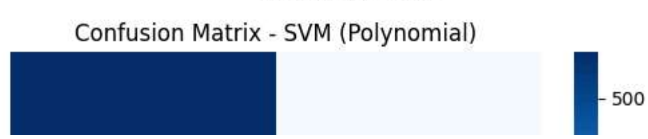

```
KNN Tree Comparison:
           Accuracy  Precision    Recall  F1 Score  Train Time (s)
kd_tree    0.907709   0.886111  0.878788  0.882434        0.024995
ball_tree  0.907709   0.886111  0.878788  0.882434        0.018309
0.9077090119435306
```

```
# -------------------------
# 9. SVM - kernels
# -------------------------
svm_params = {
    'Linear': {'kernel':'linear', 'C':1.0, 'probability':True},
    'Polynomial': {'kernel':'poly', 'degree':3, 'C':1.0, 'gamma':'scale', 'probability':True},
    'RBF': {'kernel':'rbf', 'C':1.0, 'gamma':'scale', 'probability':True},
    'Sigmoid': {'kernel':'sigmoid', 'C':1.0, 'gamma':'scale', 'probability':True}
}
```

```
svm_results = {}
for name, params in svm_params.items():
    svm = Pipeline([('scaler', StandardScaler()), ('clf', SVC(**params))])
    m, y_pred, metrics = evaluate_model(svm, X_train, X_test, y_train, y_test)
    svm_results[name] = metrics
    plot_confusion(y_test, y_pred, f'SVM ({name})')
    plot_roc(m, X_test, y_test, f'SVM ({name})')

svm_df = pd.DataFrame(svm_results).T
print("\nSVM Kernel Comparison:")
print(svm_df)
```



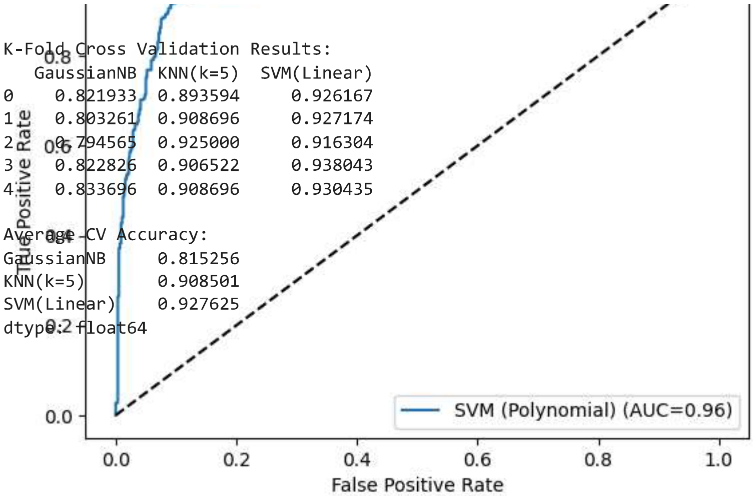


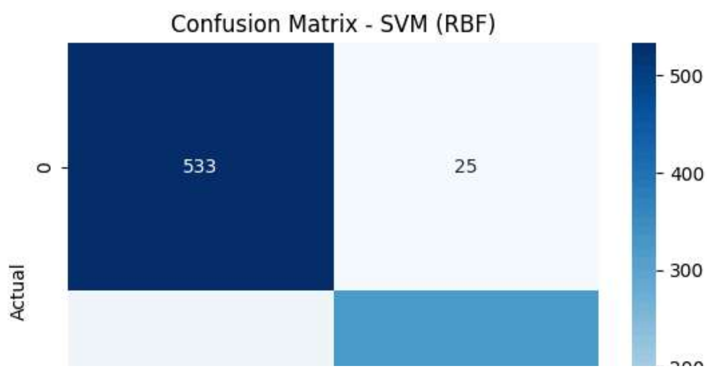



```
# --------------------------
# 10. K-Fold Cross Validation
# --------------------------
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_results = {}
models_for_cv = {
    'GaussianNB': nb_models['GaussianNB'],
    'KNN(k=5)': Pipeline([('scaler', StandardScaler()), ('clf', KNeighborsClassifier(n_neighbors=5))]),
    'SVM(Linear)': Pipeline([('scaler', StandardScaler()), ('clf', SVC(kernel='linear'))])
}
```

```
for name, model in models_for_cv.items():
    scores = cross_val_score(model, X_scaled, y, cv=kf, scoring='accuracy')
    cv_results[name] = scores

cv_df = pd.DataFrame(cv_results)
print("\nK-Fold Cross Validation Results:")
print(cv_df)
print("\nAverage CV Accuracy:")
print(cv_df.mean())
```

```
K-Fold Cross Validation Results:
   GaussianNB  KNN(k=5)  SVM(Linear)
0    0.821933  0.893594     0.926167
1    0.803261  0.908696     0.927174
2    0.794565  0.925000     0.916304
3    0.822826  0.906522     0.938043
4    0.833696  0.908696     0.930435

Average CV Accuracy:
GaussianNB     0.815256
KNN(k=5)       0.908501
SVM(Linear)    0.927625
dtype: float64
```





5 Comparision Tables

Table 1: Performance Comparison of Naïve Bayes Variants

Model	Accuracy	Precision	Recall	F1 Score
BernoulliNB	0.8899	0.8843	0.8290	0.8558
MultinomialNB	0.8950	0.9424	0.7813	0.8543
GaussianNB	0.8197	0.7001	0.9485	0.8056

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.8899	0.8551	0.8676	0.8613
3	0.8892	0.8710	0.8438	0.8571
5	0.8993	0.8828	0.8585	0.8705
7	0.8950	0.8815	0.8474	0.8641

Table 3: KNN Comparison: KDTree vs BallTree

Metric	KDTree	BallTree
Accuracy	0.8899	0.8899
Precision	0.8551	0.8551
Recall	0.8676	0.8676
F1 Score	0.8613	0.8613
Training Time (s)	0.4470	0.4058

Table 4: Cross-Validation Scores for Each Model (K = 5)

Fold	Naïve Bayes Accuracy	KNN Accuracy (k=1)	SVM Accuracy
Fold 1	0.8719	0.9034	0.9359
Fold 2	0.8935	0.9076	0.9250
Fold 3	0.8891	0.9152	0.9402
Fold 4	0.8913	0.9000	0.9337
Fold 5	0.8859	0.8935	0.9326
Average	**0.8863**	**0.9039**	**0.9335 ± 0.0050**

Table 5: SVM Performance with Different Kernels and Parameters

Kernel	**Hyperparameters**	**Accuracy**	**F1 score**	**Training time**
linear	c=10.0	8.993	8.660	0.12
polynomial	c=10.0,degree=3,gamma=scale	0.8501	0.7762	0.35
RBF	c=10.0,gamma=scale	0.9261	0.9041	0.20
sigmoid	c=10.0,gamma=scale	0.8023	0.7393	0.18

6 Observation:

- KNN with k=1 achieved the highest accuracy consistently across all folds, indicating strong capability in classifying email spam versus ham.
- Naive Bayes classifiers, particularly MultinomialNB, provided stable and competitive results, showing robustness in handling text data with varying feature distributions.
- SVM, especially with the RBF kernel (C=10, gamma=`scale`), achieved the highest overall accuracy ($\approx$ 92.7) and F1 score among all kernels, demonstrating its strong capability in handling complex, non-linear decision boundaries in text data.
- Although KNN attained better peak performance, Naive Bayes models required less training time and are more scalable for larger datasets.
- The choice between KNN, Naive Bayes, and SVM depends on the trade-off between accuracy, computational efficiency, and dataset size for the specific application scenario.

GitHub Repository: https://github.com/kawvya-mk/ml-ass3

7 Conclusion:

- The experiment demonstrated that KNN (k=1) outperforms Naive Bayes variants in certain folds, but SVM with RBF kernel consistently delivered the best overall performance in terms of accuracy and F1 score on the email classification task.
- Naive Bayes remains valuable due to its simplicity, fast training, and effectiveness on high-dimensional, sparse data typical in text classification, while SVM offers robust performance with non-linear patterns but requires slightly more computational resources.
- For deployment:
  - Naive Bayes is suitable for quick predictions on large datasets.
  - KNN is ideal when highest accuracy is needed and resources allow.
  - SVM is a balanced choice when accuracy and robustness against complex patterns are critical, with moderate training time.